NGAPP – Final Presentation

Bernhard Kausl
Armin Luntzer
Roland Ottensamer

ESTEC, Sept. 19, 2014

institut für astrophysik
UNIVERSITÄTSSTERNWARTE WIEN

Agenda

NGAPP (Next Generation Astronomy Processing Platform)

- Background and Objectives
- MPPB Hardware
- RUAG Use Case (not part of NGAPP)
- Astronomy Use Cases
- Performance Evaluation
- Introduction to a Lean Operating System for MPPB
- MPPB use
- Conclusion and presentation of key results
- Final Discussion

NGAPP – Background

NGAPP (Next Generation Astronomy Processing Platform)

- ESA contract number: 40000107815/13/NL/EL/fk
- Original NGAPP Idea:

General objective was the design of a powerful processing platform to be prepared for the use in several future (astronomy) science missions.

- Several activities have been started in recent years by ESA/ESTEC
- A set of available processing cores has been analyzed (also commercial parts), e.g., Atmel DIOPSIS, FFTC, C6701(2)x etc.
- Massively Parallel Processor Bread-boarding Study (MPPB)
  - RECORE Systems (Netherlands)
  - Timeframe: 2009-2011
  - Result prototype hardware platform based on Xilinx FPGA

NGAPP – Background

NGAPP (Next Generation Astronomy Processing Platform)

- Current Baseline/Objective:

The development of a high-performance, scalable, rad-hardened, highly integrated, mixed signal Data Processor for Sensors, Instruments, and Processing Units with excellent re-use potential for further Science & Exploration missions

→ **Scalable Sensor Data Processor (SSDP) ASIC**

- SSDP Prototypes end 2015
- FMs available ~2016/17

NGAPP – Objectives and Study Flow

NGAPP (Next Generation Astronomy Processing Platform)

- NGAPP Project Objectives at RSA / UniVie

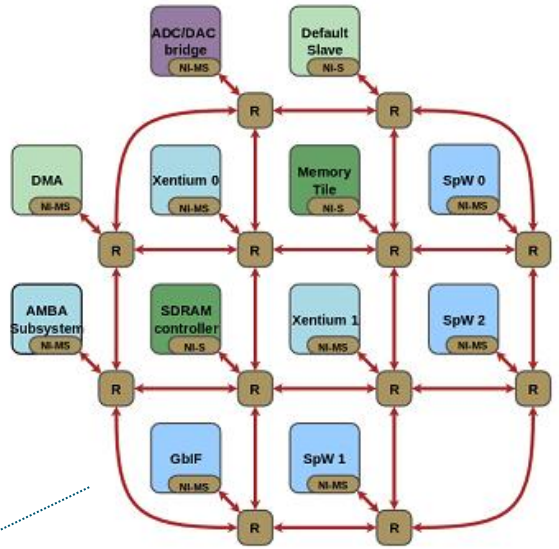
The objectives of NGAPP were to evaluate the MPPB with respect to processing performance and capabilities, aiming to evaluate applicability for possible space missions, by subjecting it to software benchmarking and architecture analysis.

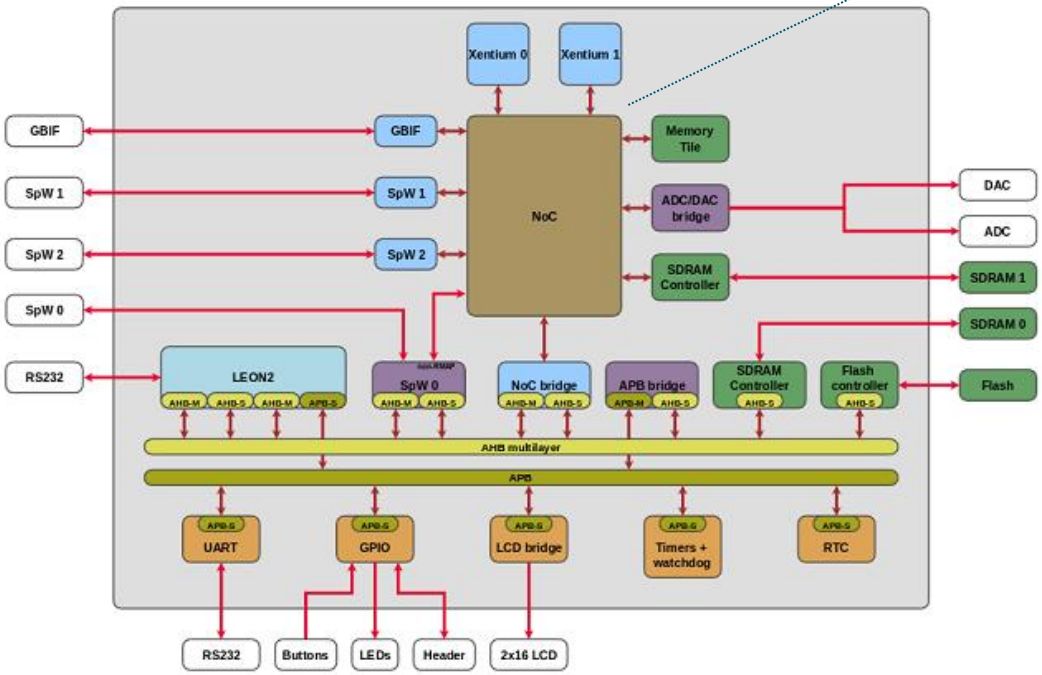
- Co-operation with University of Vienna, Department of Astrophysics
- We followed a twofold Approach
  - Software Benchmarking of MPPB Prototype
  - Analysis of MPPB features w.r.t. design of a Data Processing Unit (DPU) and applications
- Outcome
  - Inputs to SSDP ASIC specification → **Improvement Suggestions**
    - Development of the SSDP was already on-going (2013)
  - Performance software library
  - Experimental LeanOS

MPPB / SSDP Introduction

MPPB Prototype Platform

- 2 Xentium DSP cores, 1 LEON2 processor
- High-speed interfaces (SpaceWire, ADC/DAC)
- 50MHz system clock
- **Network-on-Chip (NoC)**





Together ahead. RUAG

RUAG Use Case: High Sensitivity GNSS Receiver

Motivation

- RUAG Space Austria has leading role in the development of high-performance, high-reliability GNSS receiver, for Precise Orbit Determination (POD)
  - Delivery to ESA, EC Earth Observation programs, NASA
  - Successful in-orbit heritage
- A possible use case related to SSDP are High-Sensitivity (HS) environments, refers to situation where very **low carrier-to-noise-power-density ratio C/N0 (dB-Hz)** levels expected
- General applications are e.g. indoor GNSS, i.e., non-space
- In space, similar situations are expected, e.g., in
  - GEO / GTO orbits, or e.g. in
  - GPS utilization for Moon missions
    - Rationale are free-space loss together with antenna alignment

High Sensitivity GNSS Receiver

Problem and Algorithm for fast Acquisition

- Acquisition of GNSS receiver is one of the most computation intense task
  - Cross-correlation of received GNSS signal with local code replica (Spread-Spectrum technique) + peak search. Until now, this task is done by dedicated ASICs.
- Cross-correlation in time domain can be equivalently performed in frequency domain by make use of FFT, and IFFTs and several dot-wise Multiplications (in frequency domain)
- Computation complexity reduces to $O(N \log N)$ compared to $O(N^2)$
- Remember that we are dealing now with **low C/N0 (dB-Hz)** levels
- Therefore, **long integration** is required, but there are limits!
- The algorithm combines Coherent (C) and In-Coherent (IC) integration
- A so called **double-FFT based acquisition algorithm** was introduced by [Seco-Grandados, 2012].
  - A huge amount of CFFT computations are to be processed in a given time period!

Performance Benchmarks

Performance Benchmarks for relevant rad.-hard. Processors of **Standard DSP Algorithms**

Benchmarks	LEON2 @100MHz	Xentium @100MHz	FFTC @123MHz
CFFT-64, 16-bit	97µs	1.75µs*	0,37µs*
CFFT-1024, 16-bit	2,58ms	65µs[1]	10µs[2]
CFFT-8192, 16-bit	29,38ms	486µs*	104µs*
FIR-32, 16-bit complex	2,8µs	0,16µs[1]	-

* .. log2-scaled based on benchmark results of [1] or [2]

[1] RECORE Systems, Xentium DSP product brief; FFT computation based on radix-4 algorithm
[2] FFTC-ASD-TN-003, A, August 2009

High Sensitivity GNSS Receiver

Performance Estimate and Conclusion

- We have established a numerical example considering a L1 C/A code, integration over 2 payload bits (40ms), and applied the algorithm stated before
  - Goal: Estimating the required processing power
- Hard real-time: computing a definite set of FFT (different length, power of 2, zero-padded) in a given time, e.g., **execution time < 40ms** is required!
- Course implication: ~20 Xentium cores would be required (for 1 channel only), for a the example above.
  - Just to see the orders of magnitude
- A conclusion is that we need powerful FFT rad-hard. processors / processing units for such or similar applications at RUAG Space
- The current ESA DSP roadmap gives us confidence (further developments based on SSDP, scalability and mutli-core approach)

Use Cases: Astronomic Science Missions

institut für astrophysik
UNIVERSITÄTSSTERNWARTE WIEN

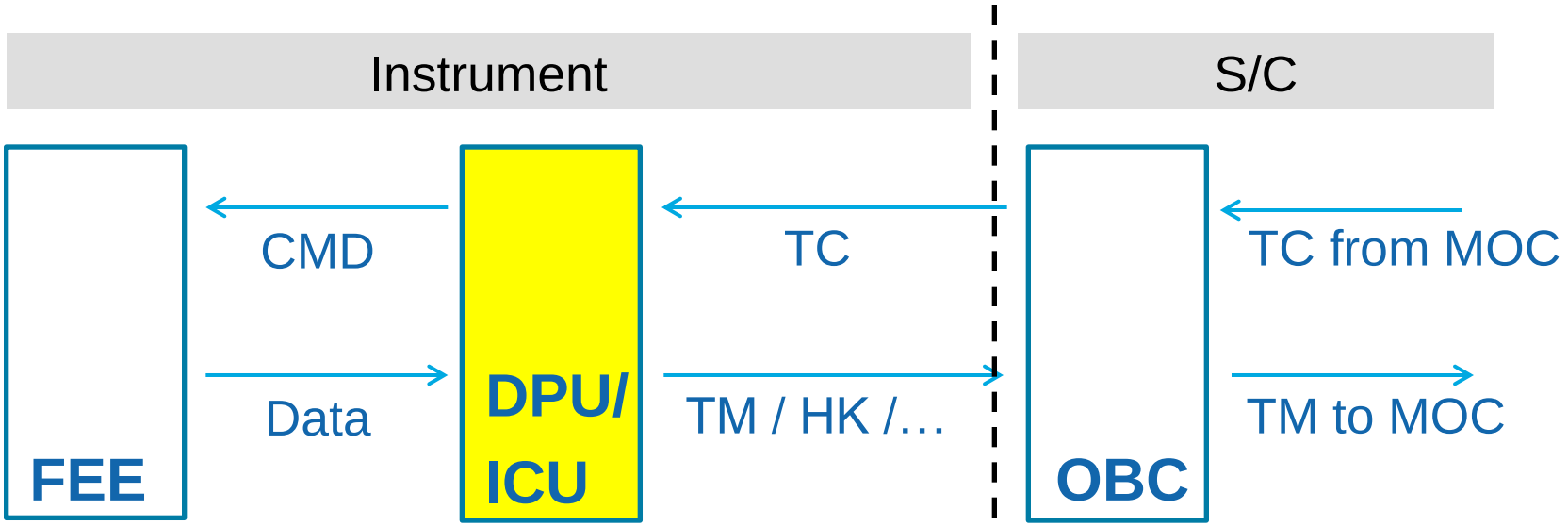
- **ESA Cosmic Vision**
  - long-term planning of science missions
  - contains S, M and L – class missions
  - ESA provides the S/C through an industrial Prime
  - Payload Instruments are provided through institutional consortia

- **Each Instrument has budgets…**
  - mass, power, volume, telemetry

- **…and defined interfaces to the S/C**
  - optical, electrical, mechanical

institut für astrophysik
UNIVERSITÄTSSTERNWARTE WIEN
Together ahead. RUAG

Payload Instrument Data Processing



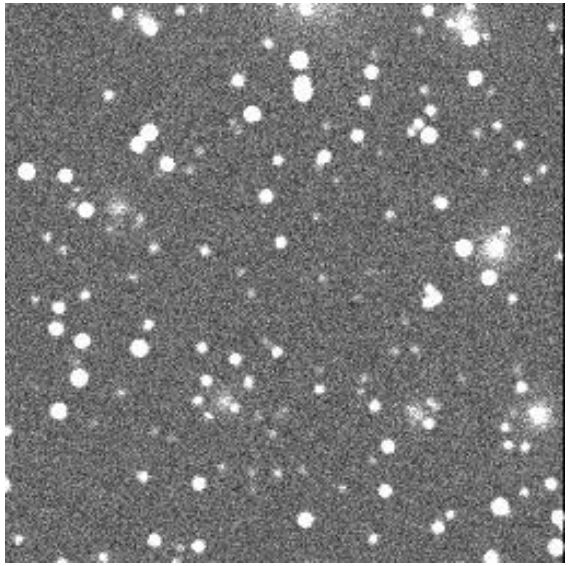
- Detector / FPU control
- Detector readout

- Instrument control
- Data processing
- HK collection
- Centroiding
- TM generation
- FDIR

- TM/TC interface
- Instrument timeline
- Mass storage

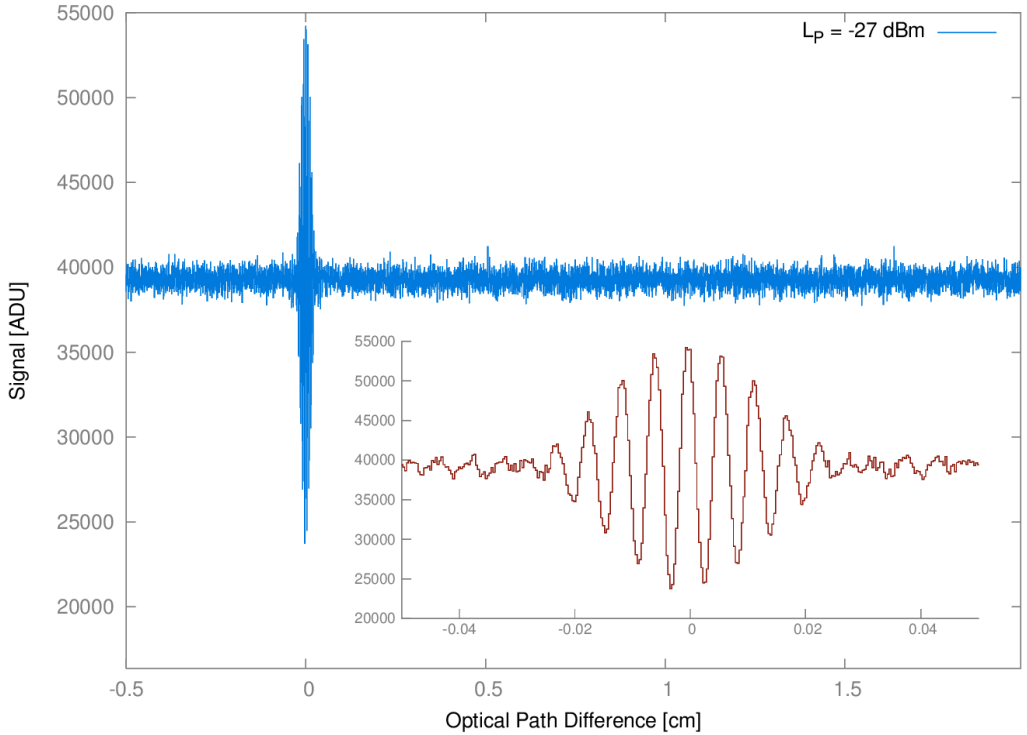
Astronomic Use Cases

- **Raw Science Data >> TM budget**
  - typically by a factor of 5 – 50
  - 100k – 1M samples/s
  - 16 bit integer data type
- **SNR is low**
  - noise is to be kept intact



X-ray data (result)

Far-IR FTS data

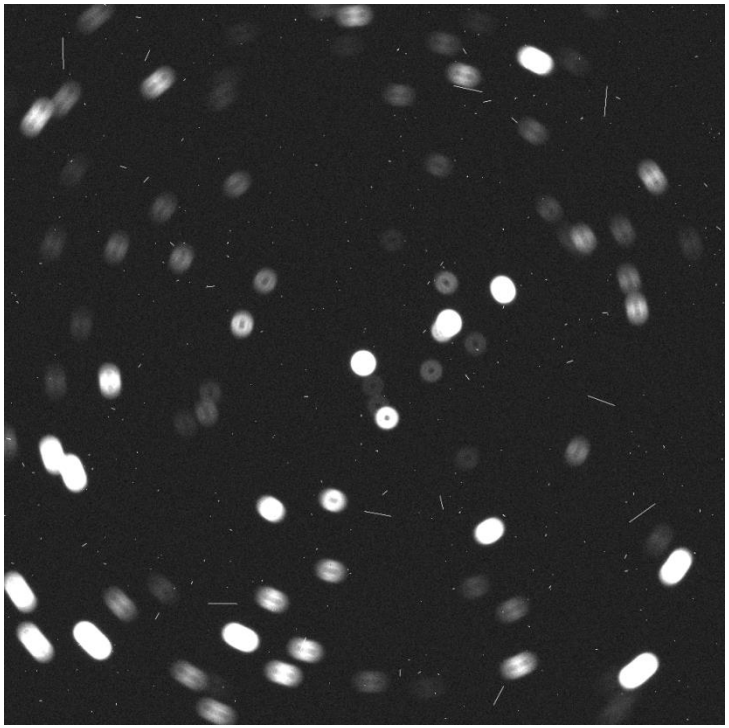


institut für astrophysik UNIVERSITÄTSSTERNWARTE WIEN

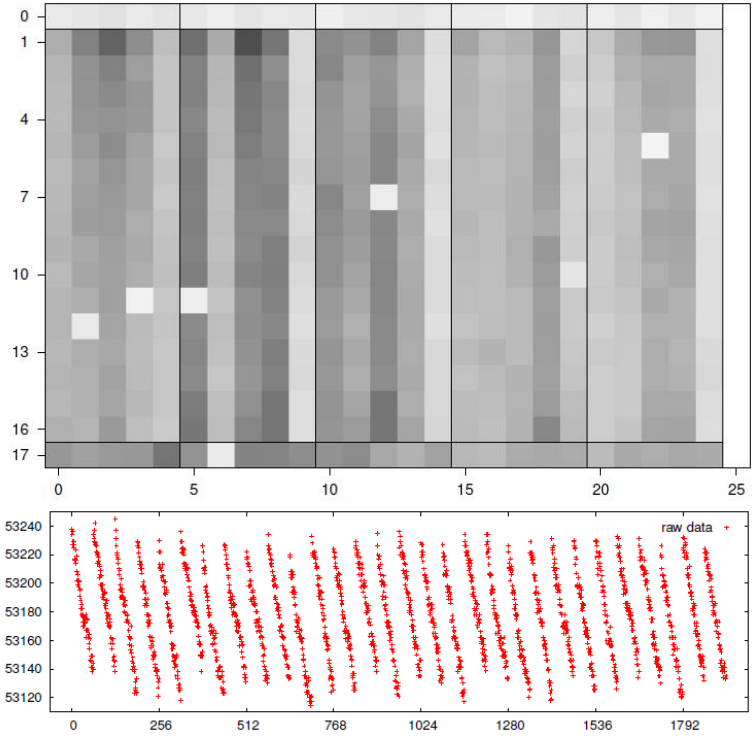
Together ahead. RUAG

Astronomic Use Cases

- **Additional Tasks**
  - ancillary data, e.g. centroiding
  - deglitching
  - on-board calibration



High-precision photometry data



Far-IR spectroscopy

institut für astrophysik UNIVERSITÄTSSTERNWARTE WIEN

Together ahead. RUAG

Astronomic Use Cases: Compression

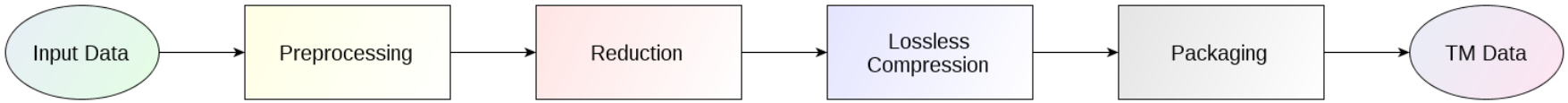
- **Lossless Steps**
  - Step 1: Reversible decorrelation or prediction
  - Step 2: Entropy Coding

- **Lossy Steps: 4 domains**
  - Temporal (e.g. stacking, decimation)
  - Spatial (e.g. windowing, binning)
  - Signal Sampling (e.g. rounding)
  - Transform (e.g. DCT + quantization)

- **For lossy steps, glitches must be controlled**

- **“The Pipeline starts in Space!”**

On-Board Data Processing Chain



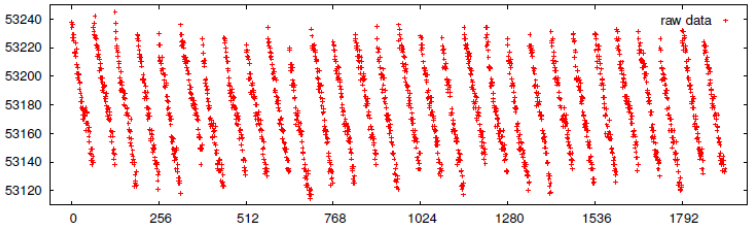
- **Data processing steps and algorithms chained together**

- **Preprocessing**
  - e.g. conditioning, integrity checks
- **Reduction**
  - lossy steps, e.g. averaging, ramp fitting
- **Lossless Compression**
  - one or 2-step entropy coding
- **Auxiliary steps**
  - all the rest, e.g. checksums

Showcase: Ramp Fitting

- **C reference implementation**

  [cycles / sample]
  - ADSP 21020: **3**
  - ARM V7 (Cortex M3): **7**
  - LEON 2: **17**
  - Xentium compiler: **6**



$$m = \Lambda \left[\left(\sum_{i=1}^{n} i \cdot x_i \right) - \frac{1}{2} (n+1) \left(\sum_{i=1}^{n} x_i \right) \right],$$

- **Xentium handwritten assembly**
  - **0.88** c/s
  - makes efficient use of LOOP feature

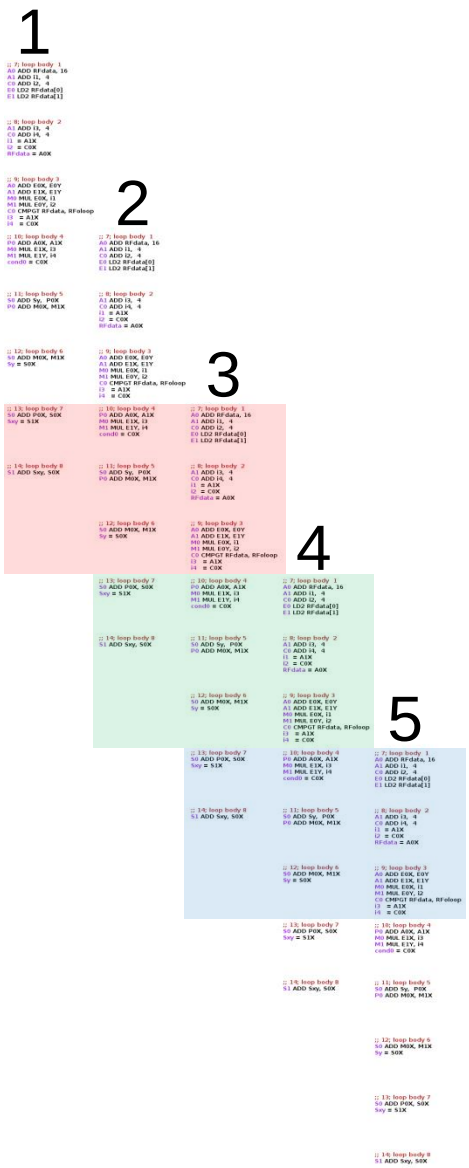
```
for (i=1; i<=n; i++)
{
      x    = data[pos++];
      Sx   += x;
      Sxi  += i * x;
}
Sxterm      =  lambda * ( (n+1) * Sx) >> 1;
slopes[r++] = (lambda * Sxi - Sxterm);
```

Ramp Fit in Xentium

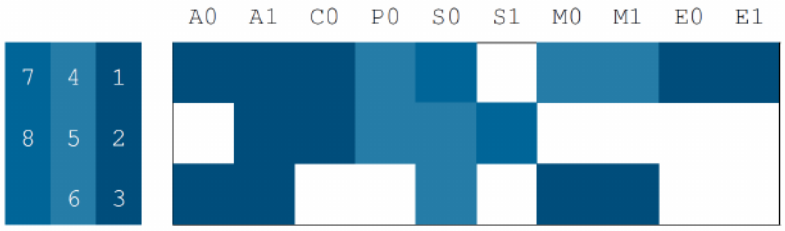
```
FastIntFixedRampFitBuffer:
      ;; 1
      A0 ADD  RFrl, 1            ; rl1
      A1 OR   0, RFdatabuf       ; RFdata
      C0 LINK
      S0 SL   RFns, 2
      cond0 = 0
      r     = 0

      ;; 2
      S0 ADD A1X,S0X
      S1 SRU RFrl, 2
      rl1      = A0X
      RFdata   = A1X
      RFretptr = C0X

      ;; 3
      RFiloop = S1X
      tmp =  S0X

.RFloop:
      ;; 4
      C0 LOOP 3, 8, RFiloop
      M0 MUL  RFrl, rl1
      A0 SUB tmp, 16
      Sy = 0

      ;; 5; loop delay slot 1
      A0 SUB 0, 3            ; i1
      A1 SUB 0, 2            ; i2
      C0 SUB 0, 1            ; i3
      RFoloop=A0X

      ;; 6; loop delay slot 2
      S1 OR 0, 0
      RFamp = M0X
      i1    = A0X
      i2    = A1X
      i3    = C0X
      i4    = 0

      ;; 7; loop body  1
      A0 ADD RFdata, 16
      A1 ADD i1,  4
      C0 ADD i2,  4
      E0 LD2 RFdata[0]
      E1 LD2 RFdata[1]

      ;; 8; loop body  2
      A1 ADD i3,  4
      C0 ADD i4,  4
      i1  = A1X
      i2  = C0X
      RFdata = A0X

      ;; 9; loop body 3
      A0 ADD E0X, E0Y
      A1 ADD E1X, E1Y
      M0 MUL E0X, i1
      M1 MUL E0Y, i2
      C0 CMPGT RFdata, RFoloop
      i3   = A1X
      i4   = C0X

      ;; 10; loop body 4
      P0 ADD A0X, A1X
      M0 MUL E1X, i3
      M1 MUL E1Y, i4
      cond0 = C0X

      ;; 11; loop body 5
      S0 ADD Sy,  P0X
      P0 ADD M0X, M1X

      ;; 12; loop body 6
      S0 ADD M0X, M1X
      Sy = S0X

      ;; 13; loop body 7
      S0 ADD P0X, S0X
      Sxy = S1X

      ;; 14; loop body 8
      S1 ADD Sxy, S0X

      ;; 15 - back in outer loop
      M0 MUL Sy,  RFamp
      M1 MUL S1X, RFrl

      ;; 16 - wait
      A1 ADD r, 1

      ;; 17
      C0 BRZ, cond0 .RFloop
      S0 SRU M0X, 1

      ;; 18
      A0 SUB M1X, S0X

      ;; 19 - BR delay slot 1
      E0 STW RFslopebuf[r], A0X
      r = A1X

      ;; 20 - BR delay slot 2
      C0 BRA RFretptr

      ;; 21+22
      NOP 2
```

```
for (i=1; i<=n; i++)
{
      x    = data[pos++];
      Sx   += x;
      Sxi  += i * x;
}
Sxterm      =  lambda * ( (n+1) * Sx) >> 1;
slopes[r++] = (lambda * Sxi - Sxterm);
```

LOOP Feature and Unit Usage




```
;; 13; loop body 7
S0 ADD P0X, S0X
Sxy = S1X

;; 10; loop body 4
P0 ADD A0X, A1X
M0 MUL E1X, i3
M1 MUL E1Y, i4
cond0 = C0X

;; 7; loop body  1
A0 ADD RFdata, 16
A1 ADD i1,  4
C0 ADD i2,  4
E0 LD2 RFdata[0]
E1 LD2 RFdata[1]

;; 14; loop body 8
S1 ADD Sxy, S0X

;; 11; loop body 5
S0 ADD Sy,  P0X
P0 ADD M0X, M1X

;; 8; loop body  2
A1 ADD i3,  4
C0 ADD i4,  4
i1  = A1X
i2  = C0X
RFdata = A0X

;; 12; loop body 6
S0 ADD M0X, M1X
Sy = S0X

;; 9; loop body 3
A0 ADD E0X, E0Y
A1 ADD E1X, E1Y
M0 MUL E0X, i1
M1 MUL E0Y, i2
C0 CMPGT RFdata, RFoloop
i3  = A1X
i4  = C0X
```



- **19 of 24 (30 incl. E) units in simultaneous operation**

institut für astrophysik UNIVERSITÄTSSTERNWARTE WIEN

Together ahead. RUAG

Special Attention: Floats

- **MPPB / Xentium has no FLP instructions**
  - SW emulation through compiler is slow
- **Several FLP functions done by hand in assembler**
  - typically, a factor 3 achieved
  - function calling overhead
- **Example: fmul**
  - out of the box: 49 cycles
  - handwritten: 13 cycles + 6 for the call

Example: fmul

```
ngapp_fmul:
        ;; 1:
        A0 XOR RA6, RB6                 ; [sign a]: sign =
        A1 ORC 0,0xFF000000             ; make 0x00ffffff
        S0 SRU RA6, 23                  ; [exponent a]: shi
        S1 SRU RB6, 23                  ; [exponent b]: shi
        P0 OR 0, 0x00800000             ; [mant b]: const 0

        ;; 2:
        A0 AND S0X, 0xFF                ; [exponent a]: &0x
        A1 AND S1X, 0xFF                ; [exponent b]: &0x
        S0 AND A0X, 0x80000000          ; [sign c]
        S1 SUB P0X, 1                   ; make c7fffff
        C0 OR RA6, c800000              ; attach 1 in front
        P0 OR RB6, c800000              ; attach 1 in front
        c800000 = P0X
        cffffff = A1X

        ;; 3:
        A0 AND C0X, cffffff             ; mantissa a ready
        A1 AND P0X, cffffff             ; mantissa b ready
        S0 ADD A0X, A1X                 ; [exponent] = ea +
        C0 LINK                         ; Get the return ad
        S1 OR 0, 0                      ; prepare return va
        P0 SUB c7fffff, 0x00400000      ; create c3fffff fr
        c7fffff = S1X
        expa = A0X
        expb = A1X
        sign = S0X

        ;; 4:
        P0 SUB S0X, bias                ; exponent is ready
        M0 MUL A0X, A1X                 ; do the 64 bit mul
        A0 OR 0, 0x00400000             ; const 0x400000
        RA6 = S1X                       ; return value = 0
        c3fffff = P0X
        return = C0X                    ; save return addre

        ;; 5:
        P0X==0 C0 BRA return            ; if exp==0 return 0
        S0 SL P0X, 23                   ; shift exponent into position
        c400000 = A0X

        ;; 6: mantissa is now available
        A0 AND M0X, cffffff             ; 24 bit rest for 48 bit result
        A1 AND M0X, c7fffff             ; 23 bit rest for 47 bit result
        S0 SL M0Y, 8                    ; prepare leading part for 48 bit mant
        S1 SL M0Y, 9                    ; prepare leading part for 47 bit mant
        P0 CMPGT M0Y, 0x00007fff        ; do we have a 48 bit result or 47?
        C0 OR S0X, sign                 ; add sign to shifted exponent
```

```
        ;; 7:
        S0 SRU M0X, 24                  ; 8 bit tail for 48 bit result
        S1 SRU M0X, 23                  ; 9 bit tail for 47 bit result
        A0 AND S0X, c7fffff             ; 23 bit mantissa
        A1 AND S1X, c7fffff             ; 23 bit mantissa
        P0X!=0 P0 ADDU c800000, C0X     ; conditionally increment expa for 48 bit result
        cond48 = P0X
        expa = C0X                      ; save the sign combined exponent
        rest24 = A0X
        rest23 = A1X

        ;; 8: also start with final rounding as in LLVM mulsf3.c
        expa = P0X                      ; conditionally save new exponent
        C0 CMPGT rest24, c800000        ; if (productLo > signBit) ... productHi++;
        P0 CMPGT rest23, c400000
        A0 AND S0X, 1           ; pHi24 & 1
        A1 AND S1X, 1           ; pHi23 & 1
        pHi24 = S0X
        pHi23 = S1X
        pLo24 = A0X
        pLo23 = A1X

        ;;  9:
        S0 ADD pHi24, C0X
        S1 ADD pHi23, P0X

        ;; 10: start combine
        ;; note: by using ADDU instead of OR we automagically do the following rounding step:
        ;;       if (productLo == signBit) productHi += productHi & 1
        S0 ADDU S0X, pLo24              ; combine hi and low part 48
        S1 ADDU S1X, pLo23              ; combine hi and low part 47

        ;; 11:
        C0 BRA return                   ; --> RETURN with 2 delay slots
        A0 OR S0X, expa                 ; form 48 bit result

        ;; 12: DS1
        RA6 = A0X                       ; store 48 bit result
        cond48==0 A1 OR S1X, expa       ; conditionally form 47 bit result

        ;; 13: DS2
        RA6 = A1X                       ; return 47 bit result depending on last condition
```

- **Half the time is spent with datatype separation/build**

Benchmarks: Selection [cycles / sample]

	21020	ARM7	LEON2	XENT.	XENT. ASM	Remark
Bitstream	44 (ASM)	58	57	56	18	Compiler OK ASM x3
Mersenne Twister	84	46	57	40	-	Compiler OK
3-Tap IWT	19	57	67	24	-	DMA OK
RampFit	3	7	17	6	0.88	Theoretically 0.75
Different.	-	-	-	10	0.5	Theoretically 0.25
Transpose	2	25	14	4	1 (SIM) 11 (MPPB)	DMA penalty

System Benchmarks and Lean OS

- Data processing must be done in small TCM...

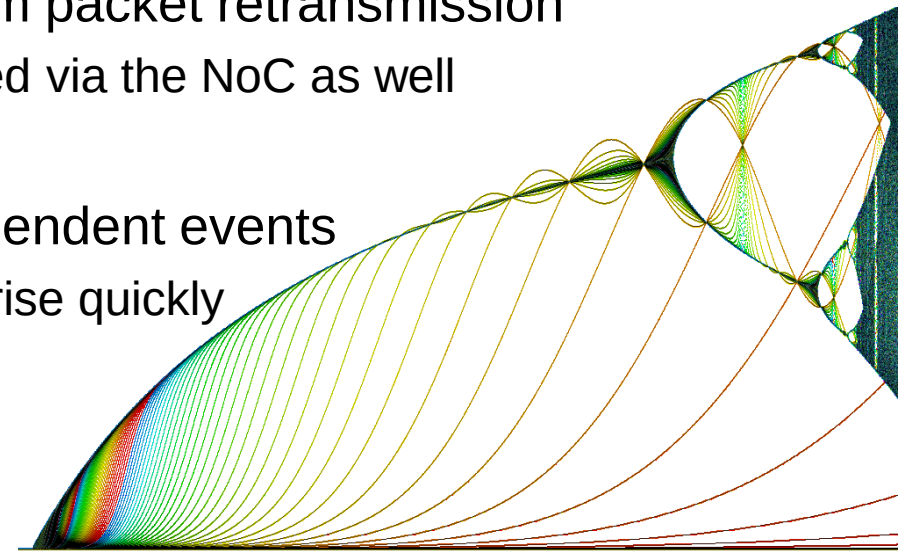
How to split a large DP task to small units
using the DMA for data I/O?

→ We created a lightweight operating system, to be able to carry out the system performance tests!

- OS considerations
  - multiple independent NoC clients generate asynchronous events
  - signal path lengths differ between all source and target nodes
    - behavior becomes “chaotic” quickly
  - DSP instruction cache memory is limited

Asynchronous events?

- all NoC attached devices can directly or indirectly generate interrupts
- events are created externally (e.g. SpW) and internally (e.g. DMA)
- NoC packet transmission times depend on routes and current load
  - completion will usually generate another event
- network interfaces can suffer from packet retransmission
  - network packets will be forwarded via the NoC as well
- many simple, but non-linear, dependent events
  - complex, chaotic behavior will arise quickly



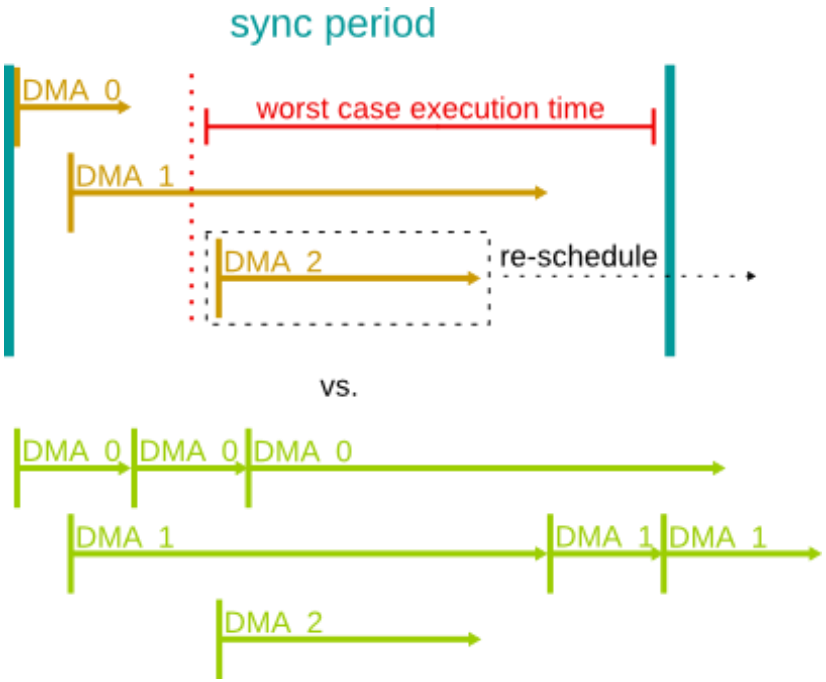
Determinism?

rigid, hard synced (periodically triggerd) setups waste up to $\sum t_{wcet}$ cycles per period

dynamic, event driven response ensures efficient resource usage, while execution time is no worse than t_{wcet}

dynamic implementations do not require reconfiguration for modified setups

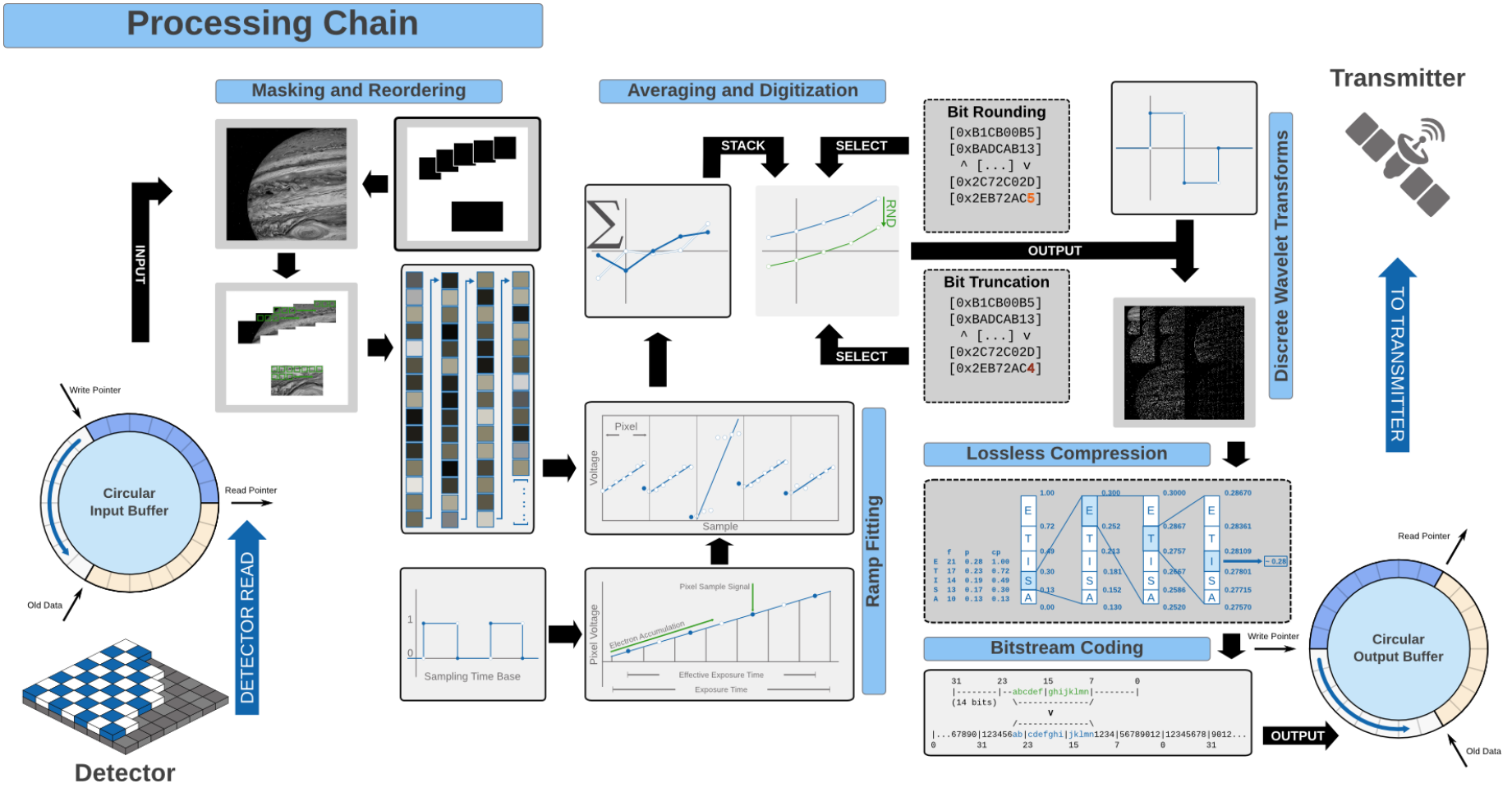
predictability is still there, just not in the global scheme



What about Real-Time?

- remember: „(hard) real-time“ does not equal „static structure“
- for on board science data processing hard time constraints can be relaxed through buffering
- if hard real-time control tasks MUST be executed on the DPU, the GPP (LEON) can take care of that as per usual
  - the processing pipeline will be fine, as long as there is enough memory to buffer incoming data

A typical On-Board Processing Pipeline



institut für astrophysik
UNIVERSITÄTSSTERNWARTE WIEN

Together ahead. RUAG

Xentium Programs

classic approach:

- single monolithic program running on the DSP
- Kernel loaded once during boot
- processing pipeline created from sequential function calls/operations

pros:

- Take code from PC and add DMA transfers for I/O

cons:

- codes size of complex pipelines can exceed i-cache size quickly
  - code must be re-fetched via the NoC every time the pipeline cycles
- even small changes/updates require full re-evaluation
  - high maintenance effort!

Xentium Programs in the LeanOS

alternative:

- multiple tiny programs (kernels), one for each functional pipeline step
- kernels should perform only one task, without any knowledge of their environment

pros:

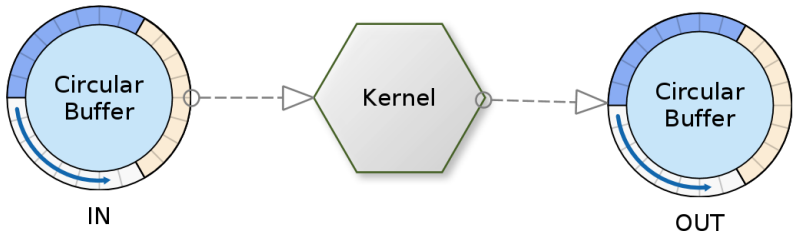
- small code size will always fit the i-cache (split into sub-kernels if not)
  - code must still be fetched via the NoC, but only once per execution cycle
- pipeline can be created from independent building blocks
- dynamic (full) resource usage, **scales with # of cores automatically**
- changes/updates affect isolated components only
  - easy maintenance
- many individual, but simpler („dumber“) units generally have less execution paths and are thus easier to trace for WCET

cons:

- Workaround for on-the fly kernel-exchange in MPPB needed

Xentium Programs: connecting the dots

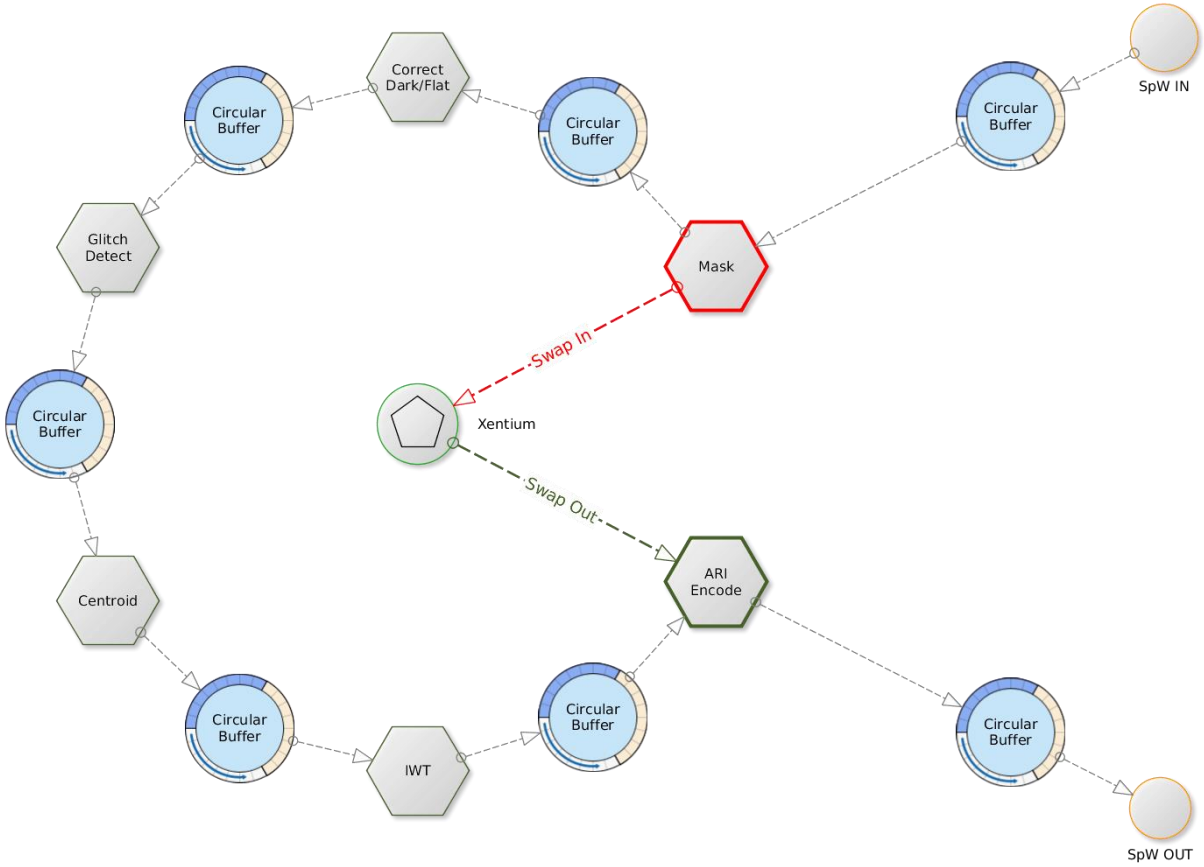
- data must be passed between kernels for processing
- using a single buffer isn‘t a good idea
  - kernels can‘t be dumb
  - no flexible scheduling



the solution: connect kernel via circular buffers!

- easy creation of a pipeline, just assign buffers
- one kernel‘s output buffer is another kernel‘s input buffer
- circular buffers are inherently accessible from multiple DSPs
  - „funnel“ the final data product or split input into separate streams
  - data-parallel execution on multiple Xentiums for throughput-critical sections

Xentium Programs: closing the loop

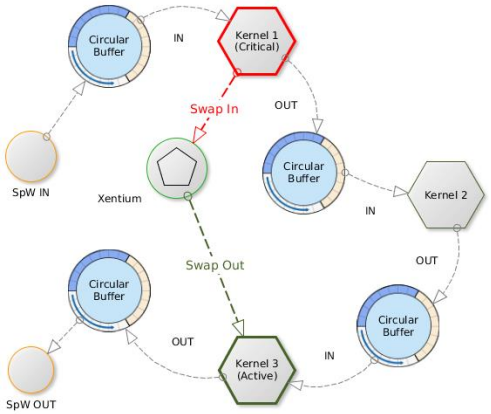


institut für astrophysik UNIVERSITÄTSSTERNWARTE WIEN

Together ahead. RUAG

Xentium Programs: Scheduling

kernels must be scheduled for execution by some metric:



circular buffer fill levels!

- define a critical input buffer fill level per kernel
- if buffer becomes critical, swap executed kernels
- also serves as a DSP load measure:

$$p_{load} = \frac{1}{freq} \sum_{kernels} (cycles/sample)_i \cdot samples_i$$

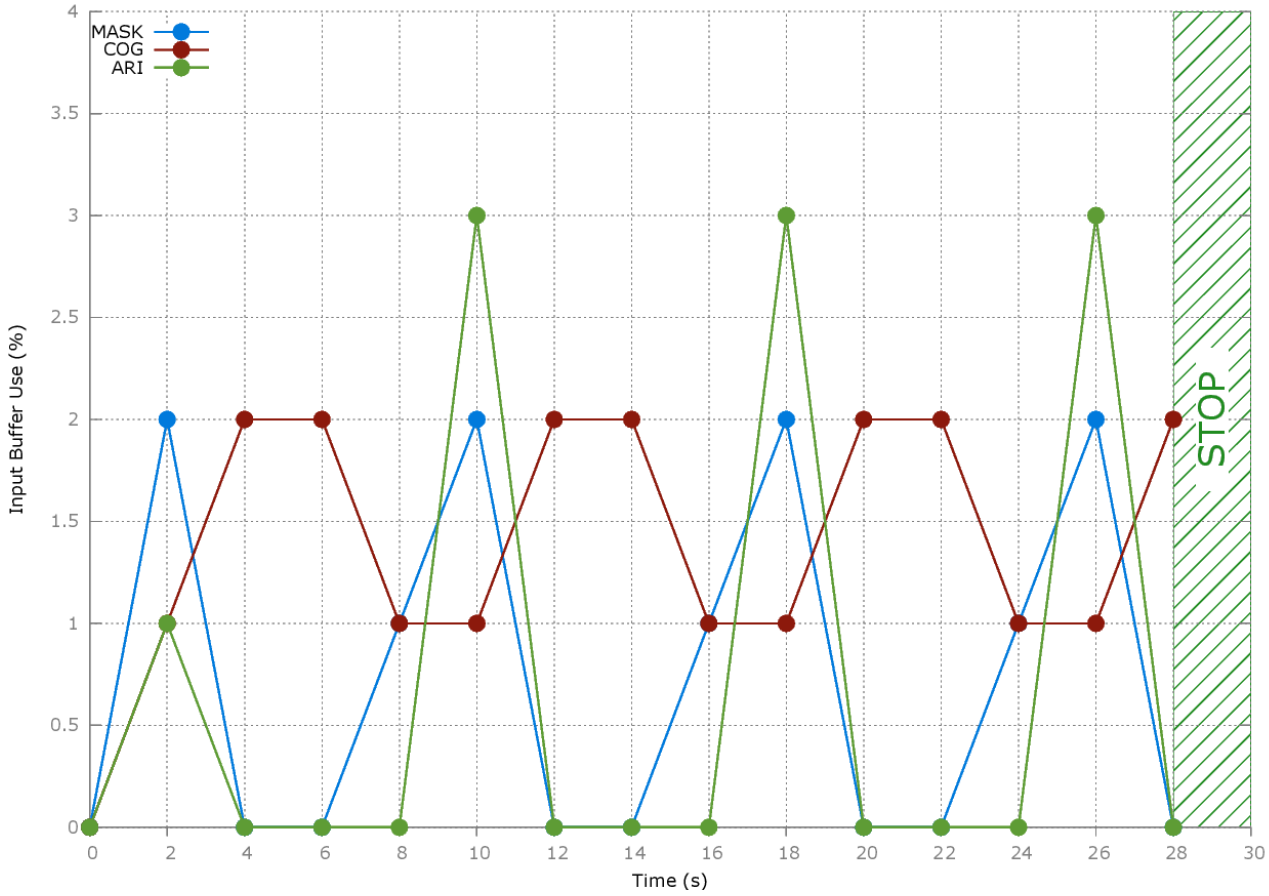
Xentium Programs: Scheduling

- self-balancing system for optimum resource usage
  - can be further improved by automatic tuning of criticality levels
- the complex interaction of NoC components and data transfers are condensed into a single value

Major difference from a fixed scheme: data accumulates and forward-propagates driven by required computational time of individual pipeline stages, not by fixed input sizes!

- kernels can be attached to dedicated Xentiums or be freely assigned
- fast stages can accumulate lots of data in their input
  - smaller number of kernel switches lead to greater efficiencies

System Test Results



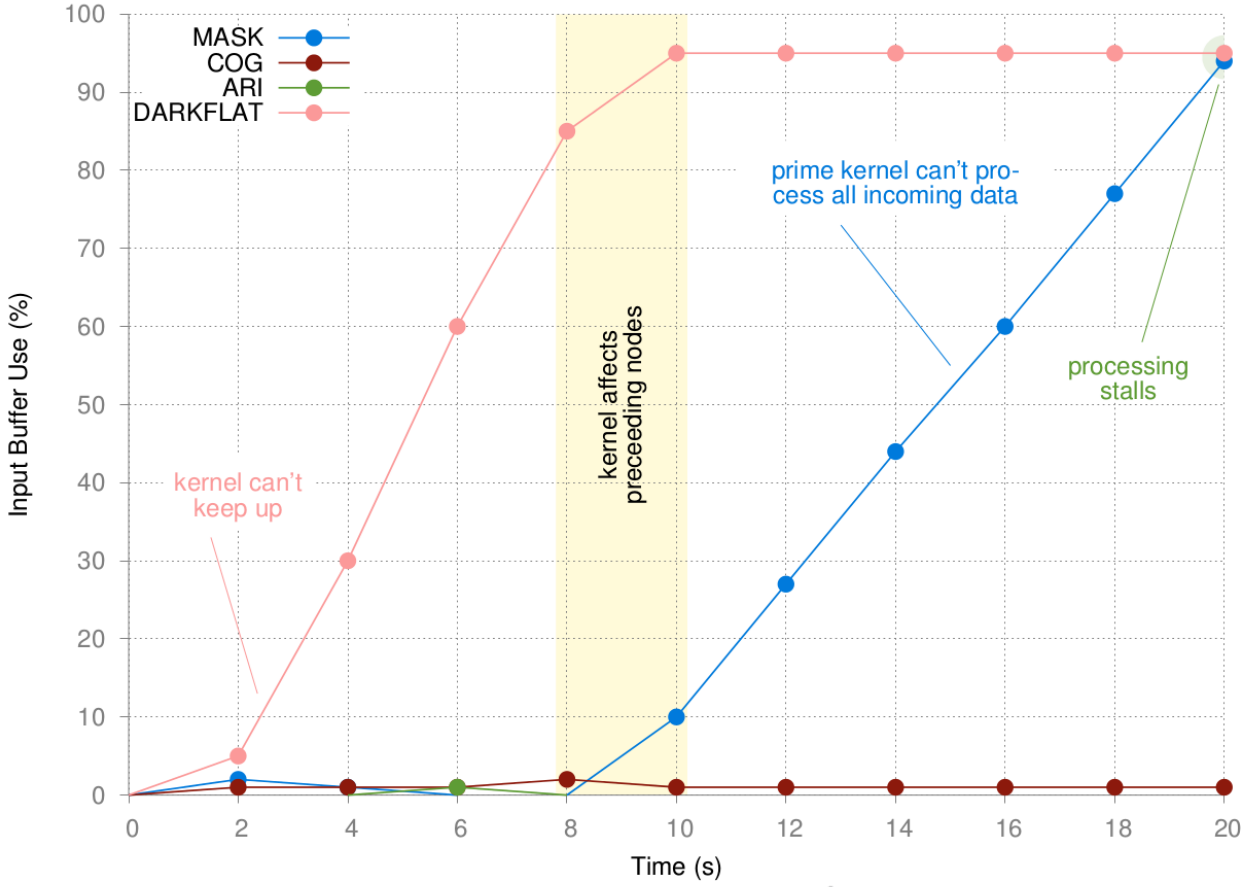
Results of a DP chain

KERNEL	PERF (c/s)	SAMPLES (IN)	LOAD (c/sec)	LOAD (%)
MASK	3	4.2M	12.6M	25.2
REORDER	11	262k	2.9M	5.8
RAMPFIT	9	262k	2.4M	4.7
DEGLITCH	39	32k	1.3M	2.6
DARK/FLAT CALIBRATE	16	32k	0.5M	1.0
DECORRELATE/ MAP/ RICECODE/ BITSTREAM	256	32k	8.4M	16.8

- 512x512 pixels input frames, temporal stack: 8x
- 2x64 Mbit/s effective data rate via SpW links (16 Hz “readout rate”)
- easily handled by a single Xentium
  - load: 56% (~1/4 of the available 2 DSP resources)

LeanOS is still experimental!

- It was created to run the performance tests, thus it is lacking features
- Memory speeds can be problematic



institut für astrophysik
UNIVERSITÄTSSTERNWARTE WIEN

Together ahead. RUAG

Key Improvement Suggestions

DMA and TCM as the key performance drivers

- A crucial **DMA** feature to be included – 2D Strides!
- The DMA does not appear to detect or report errors; no interrupt is sent if the transfer clearly failed.
  - Appear to fail or get stuck if the load on memory (SDRAM) bridges is high
- DMA transfers on the same memory block (i.e., NoC SDRAM → NoC SDRAM) are not possible.

- The **TCM** memory size should be at least 64kByte to improve performance while reducing DMA transfers and requests to the LEON2.
  - We recommend that it would be better to increase TCM size and drop the on-chip NoC SRAM, since NoC SRAM is not really useful

Key NGAPP System Requirements

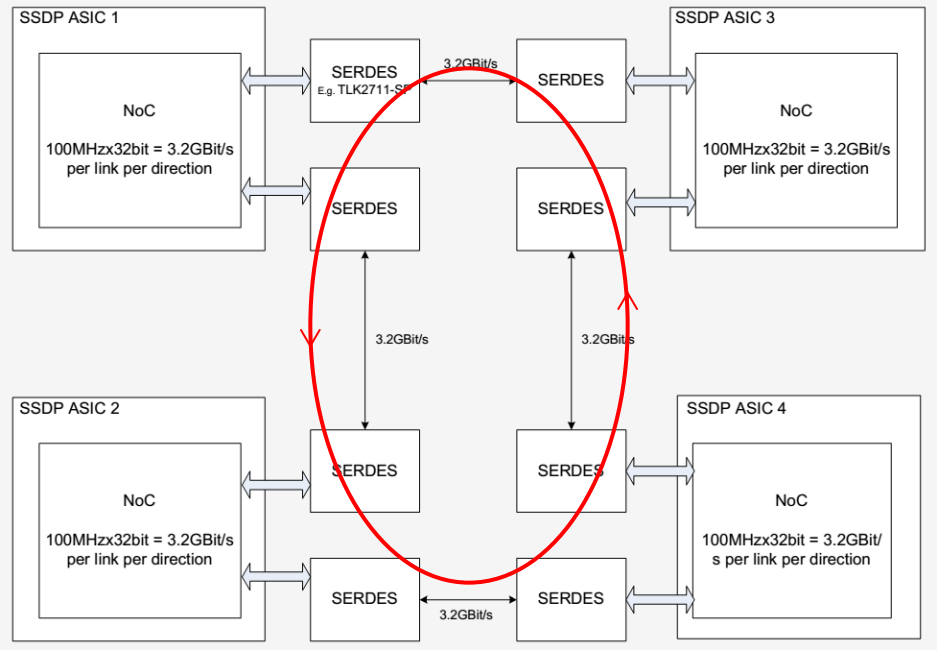
NGAPP Data Processing Unit (DPU)

	Requirement
Performance	▪ > 1000 MOPS and 500 MFLOPS (MAC operations) ▪ > Floating support with double-precision (64-bit) ▪ > CFFT-1024 below 2,4µs (RSA need)
Digital Interfaces	▪ Connection to a SpaceWire network with 3x100MBit/s (+ RMAP), 1x10MBit/s for TC/TM, bi-directional communications ▪ At least 1 UART
Analogue Interfaces	▪ ADC/DAC with at least 30MSamples/s @ 10-bits ▪ Goal: 2 ADC/DAC ports (complex signals)
Memories	▪ SDRAM up to 512MByte (payload data only), throughout rate TBD ▪ EEPROM/MRAM of at least 8MByte, 128kByte secured for boot-code ▪ Ext. memories with EDAC (single-bit error correction, double-bit error detection)
Design	▪ ITAR freeness (IP and design) ▪ Scalability (on-board) for processing power improvements
System availability (behav. without EDAC)	▪ 1 SEE related outage period per 3 days in L2 orbits ▪ Maximum recovery time after outage of 10 seconds
Power Consumption	▪ < 20W on DPU basis ▪ < 6W on IC parts basis, e.g., ASIC *Comment: Cooling mechanisms in space*
Environmental	▪ > 50kRad (Si) Total Ionizing Doses (TIDs), goal: 100kRad (Si) ▪ -20°C to 75°C operational temperature (qual.)

Key Improvement Suggestions

DPU Design – Increasing total DPU Processing Performance

- A fast interconnect between several SSDP ASICs for on-board scaling can improve total DPU performance.
  - I/Fs to external SERDES components, ideally with NoC speed (bi-directional)
  - E.g., DPU in a Ring-Topology (two I/F channels necessarily)



Together ahead. RUAG

Key Improvement Suggestions

SSDP / MPP Architecture

Component	Improvement Suggestion
Floating-Point hardware Support (FPU)	▪ Xentium Core (preferred), ▪ a node on NoC ▪ on LEON2 side
DMA	▪ Supporting 2D-strides (e.g. matrix multiplication) ▪ Number of priority levels scaled with number of Xentium cores ▪ Memory source addressing also on byte, half-word level ▪ Transfer also inside a (physical) memory (e.g. SDRAM) ▪ Bit-reversal addressing by DMA (e.g., for FFT)
TCM	▪ > 64kByte to improve performance by reduce DMA transfers
NOC SDRAM	▪ Attaching more than one SDRAM bridge to the NoC (ideal, once per Xentium core)
AMBA RAM (LEON)	▪ Required (SDRAM/SRAM), size at least 32MByte
ADC/DAC interfaces	▪ FIR filter with at least 32-taps + Decimation ▪ 2 ports supporting complex signal processing (I, Q) ▪ DC Offset compensation mechanism

Key Improvement Suggestions

SSDP / MPP Architecture

Component	Improvement Suggestion
Xentium	Support for Bitstreams ▪ “FDEP” (21020) or “BFI” (ARM) equivalent instruction
Xentium	1-cycle output initialization for M0/1 units ▪ e.g. M0 OR …

Conclusion and Outlook

NGAPP (Next Generation Astronomy Processing Platform)

Conclusions

- System/Architecture has great potential
- Approach permits scalability, continuing further developments
- Design of DPU becomes much simpler
  (lot of functionality merged into ASIC)
- powerful instruction set: LOOP construct and conditional execution
- Intuitive Assembly syntax
- Even if the DMA needs an upgrade, we HAVE a DMA!
- Not optimal for FLP applications

Conclusion and Outlook

NGAPP (Next Generation Astronomy Processing Platform)

Outlook

- SSDP is a potential candidate for future RUAG related activities, keyword: “Techniques for HSGNSS” study
- LeanOS development up to qualifiable level proposed for national funding. Development starting Q1/2 2015
- Library of Data Processing Kernels, proposal for 2015-2016
- SSDP is a potential candidate for ATHENA-WFI ICU
- Apply as candidate for SSDP beta testing 2015

Thank you for your attention!